



# Programación de Microservicios con Spring Boot y Red Hat SSO

Nivel Avanzado

Instructor: Carlos Carreño  
ccarrenovi@gmail.com



# Modulo 1. Introducción a los Microservicios

**Objetivo:** Comprender qué es la arquitectura de microservicios y cuándo aplicarla.

Duración: 1h



# ¿Qué es un microservicio? Comparación con el modelo monolítico.

- Según **ChatGPT**

“Un **microservicio** es una **unidad pequeña, autónoma y enfocada** dentro de una aplicación más grande, que se encarga de realizar una **funcionalidad específica** de negocio y que puede desarrollarse, desplegarse y escalarse **de forma independiente**”.



# Características clave de los Microservicios

## **Descomposición funcional**

- ✓ Cada microservicio implementa una función o proceso de negocio específico.

## **Despliegue independiente**

- ✓ Se pueden desplegar, escalar o actualizar sin afectar a otros servicios.

## **Autonomía**

- ✓ Cada servicio es autónomo con su propia lógica, base de datos y ciclo de vida.



# continuación

## **Tecnología heterogénea**

- ✓ Pueden ser contruidos con diferentes lenguajes, frameworks y tecnologías.

## **Comunicaciones ligeras**

- ✓ Interacción mediante API REST, gRPC o mensajería asíncrona (Kafka, RabbitMQ).

## **Escalabilidad específica**

- ✓ Permite escalar sólo los servicios que lo necesiten, no toda la aplicación.

## **Resiliencia**

- ✓ Fallos en un servicio no deben impactar a los demás (uso de patrones como Circuit Breaker).



# continuación

## Equipos pequeños y autónomos

- ✓ Cada equipo puede desarrollar, probar y desplegar su microservicio de extremo a extremo.

## Implementación continua (CI/CD)

- ✓ Permiten integración y entrega continua gracias a su naturaleza independiente.

## Observabilidad

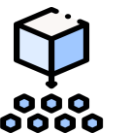
- ✓ Requieren monitoreo, logs distribuidos y trazabilidad (con ELK, Prometheus, Zipkin).

## Seguridad independiente

- ✓ Autenticación y autorización específica por microservicio si es necesario.



# Resumen



# Beneficios y desafíos de los microservicios.

## Beneficios

- ✓ Escalabilidad por componente.
- ✓ Alta disponibilidad (fallas localizadas).
- ✓ Flexibilidad tecnológica.
- ✓ Aceleración del desarrollo por equipos independientes.
- ✓ Menor riesgo al desplegar cambios.

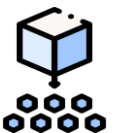




# Beneficios y desafíos de los microservicios.

## Desafíos

- ☐ Complejidad operativa y de red.
- ☐ Requiere buenas prácticas de monitoreo, trazabilidad y CI/CD.
- ☐ Coordinación de datos entre servicios.
- ☐ Latencia y tolerancia a fallos en la comunicación entre servicios.

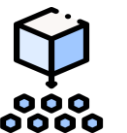
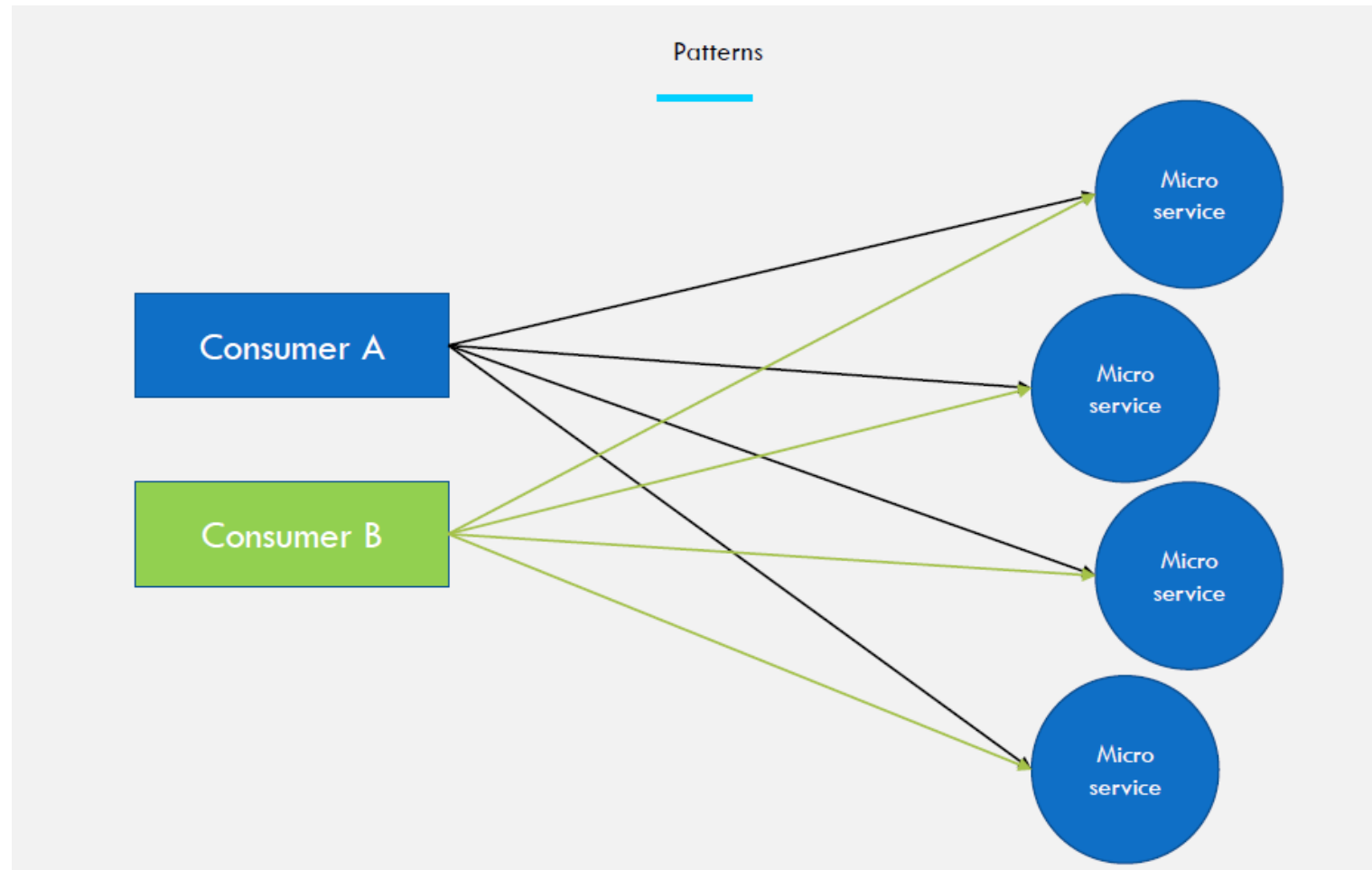


# Patrones de Diseño de Microservicios

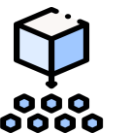
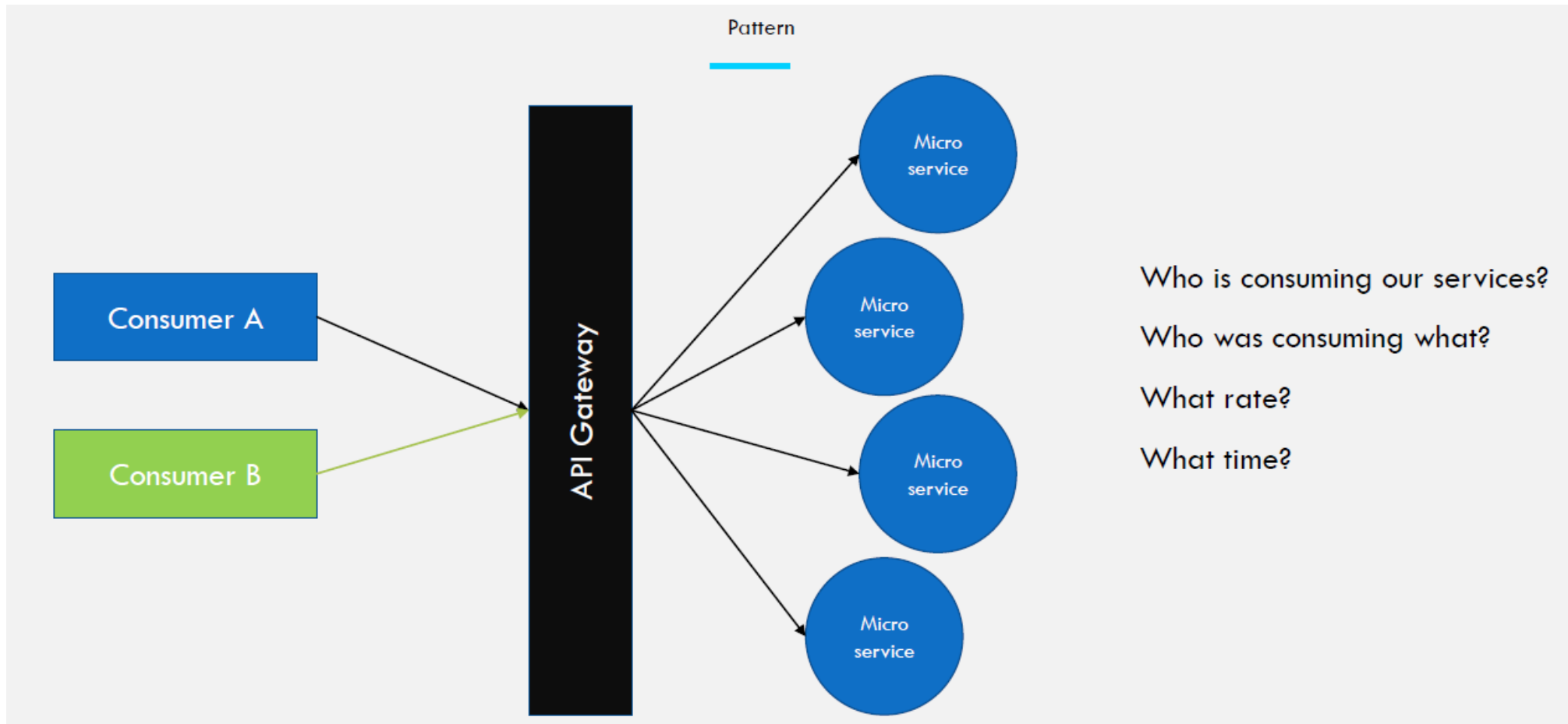
- Direct Communication
- API Gateway
- Message Broker
- Discovery (Registry Service)
- Centralized Configuration
- Retry
- Circuit Breaker
- Health Check API



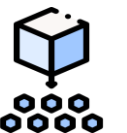
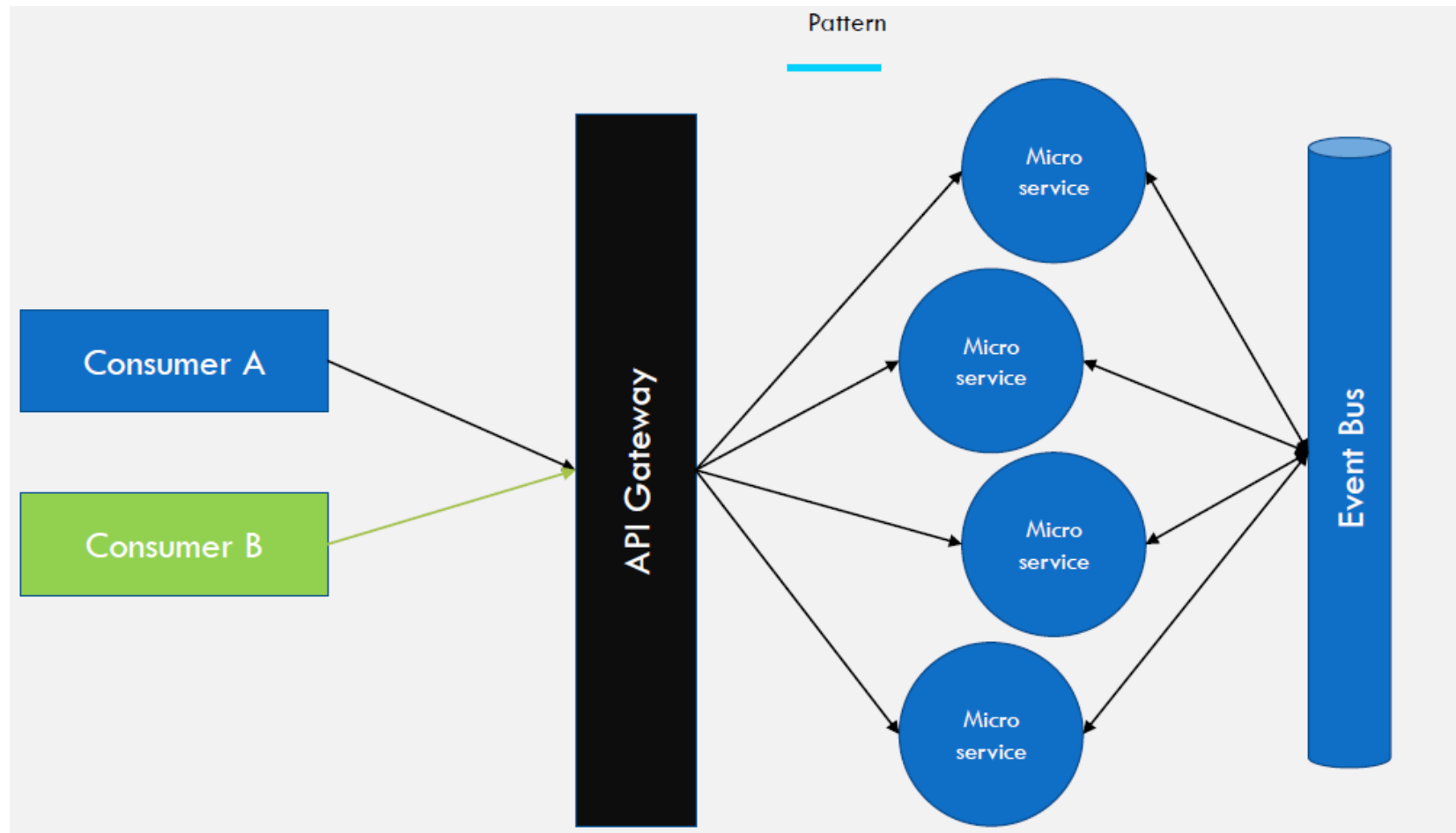
# Direct Communication



# API Gateway

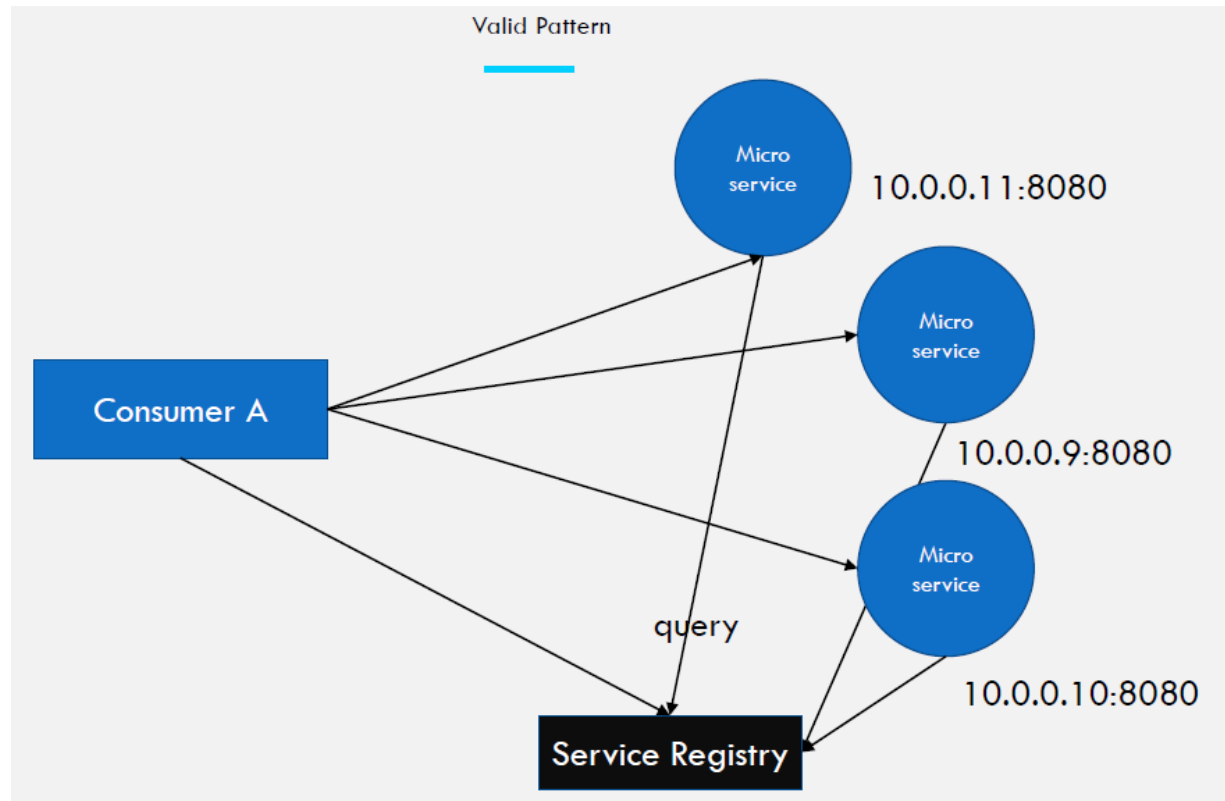


# Message Broker



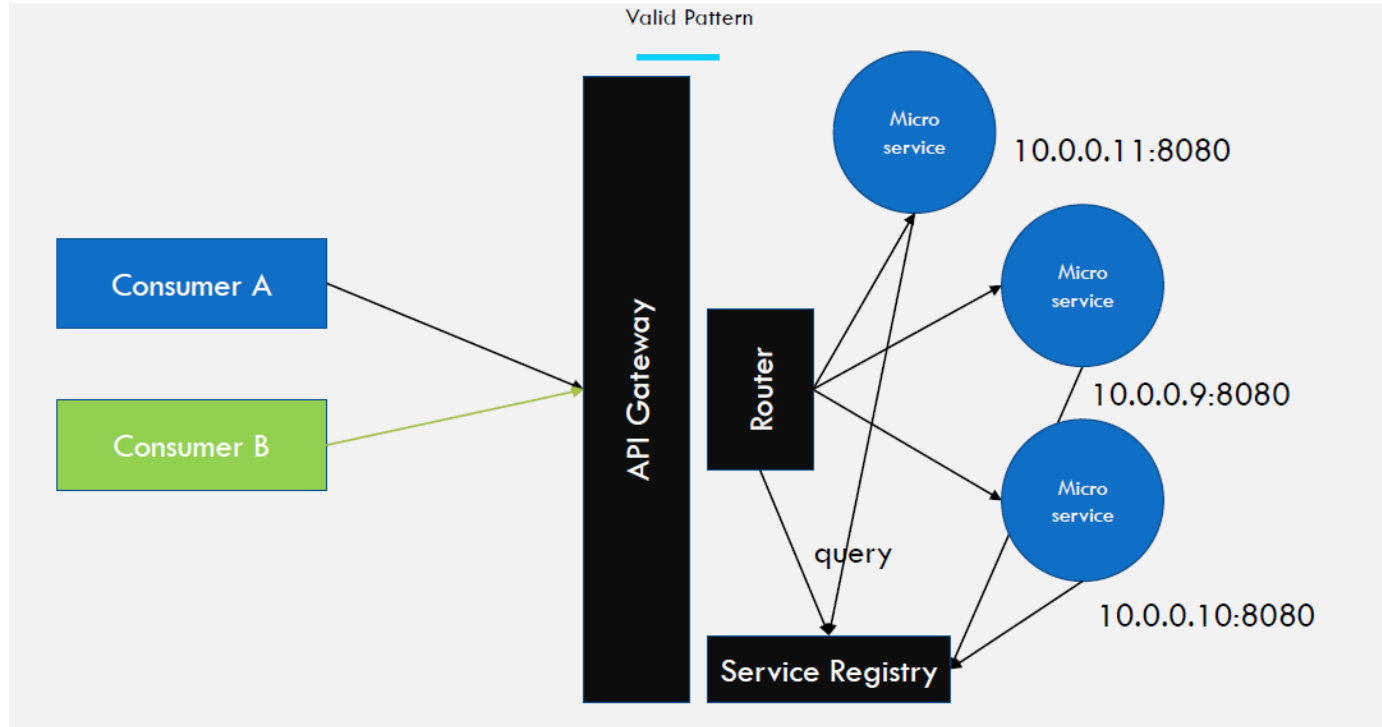
# Discovery

- Client-Side Discovery

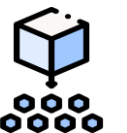
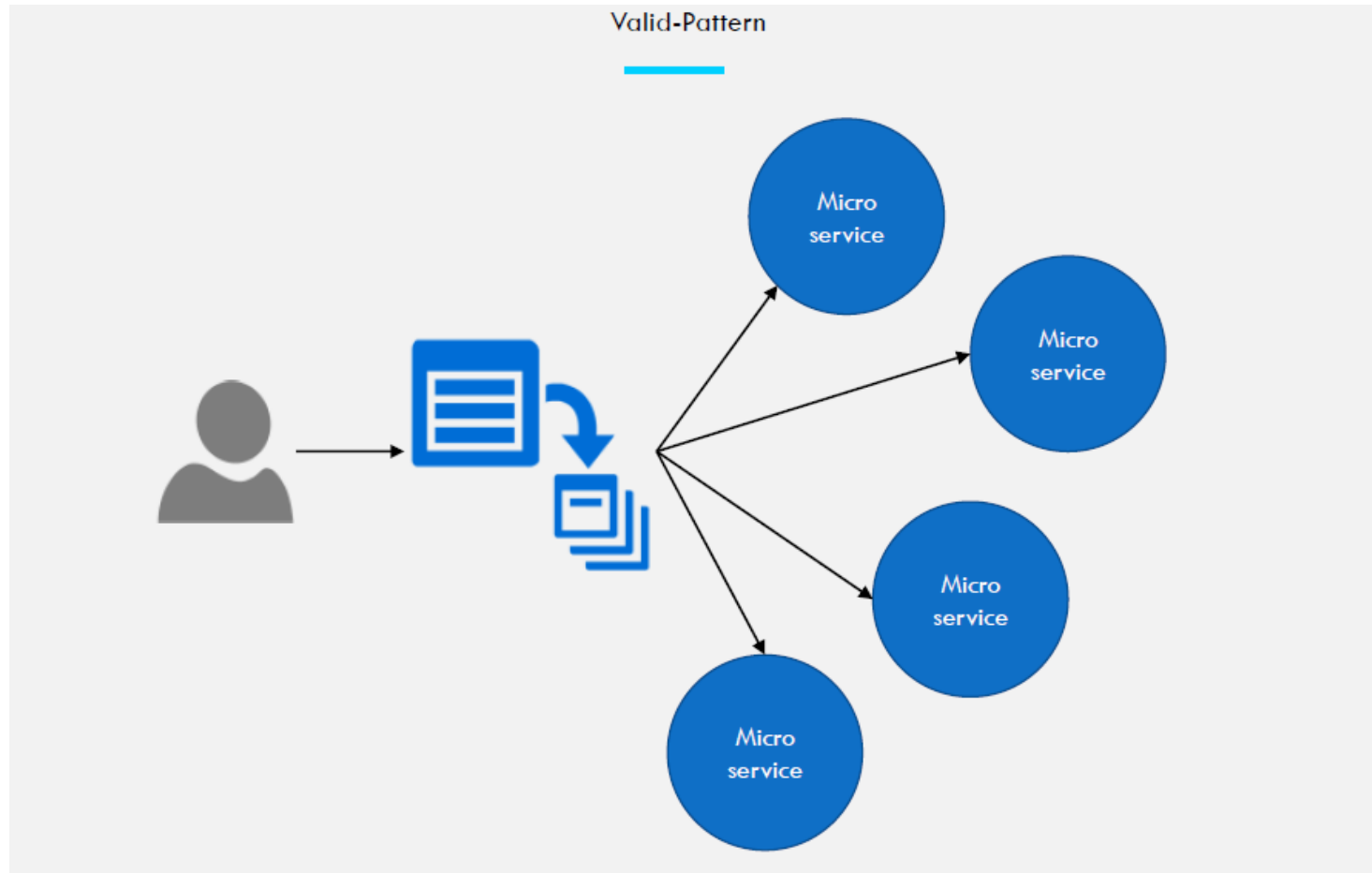


...

- Server-Side Discovery

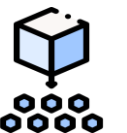
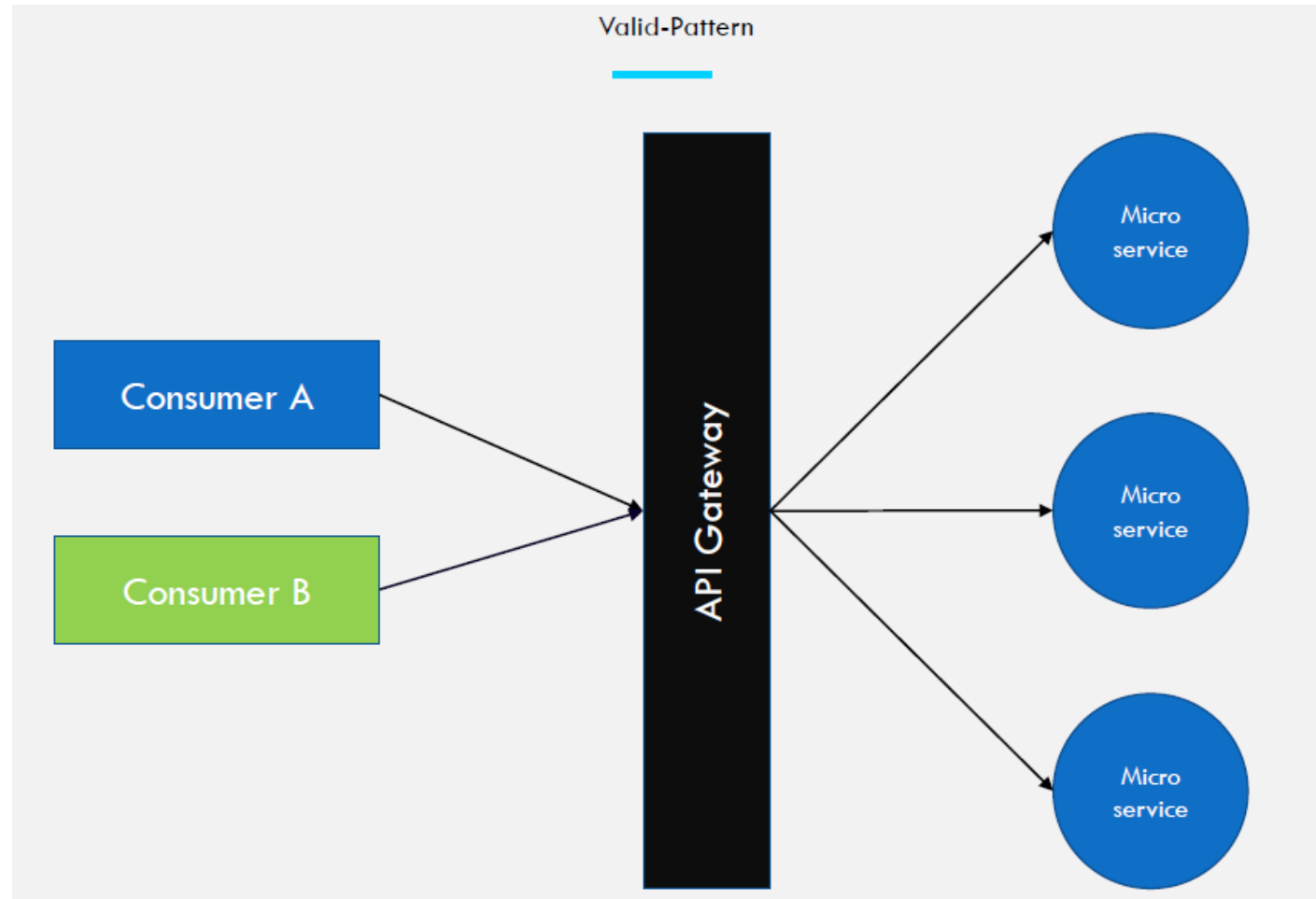


# Centralized Configuration

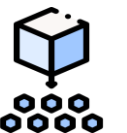
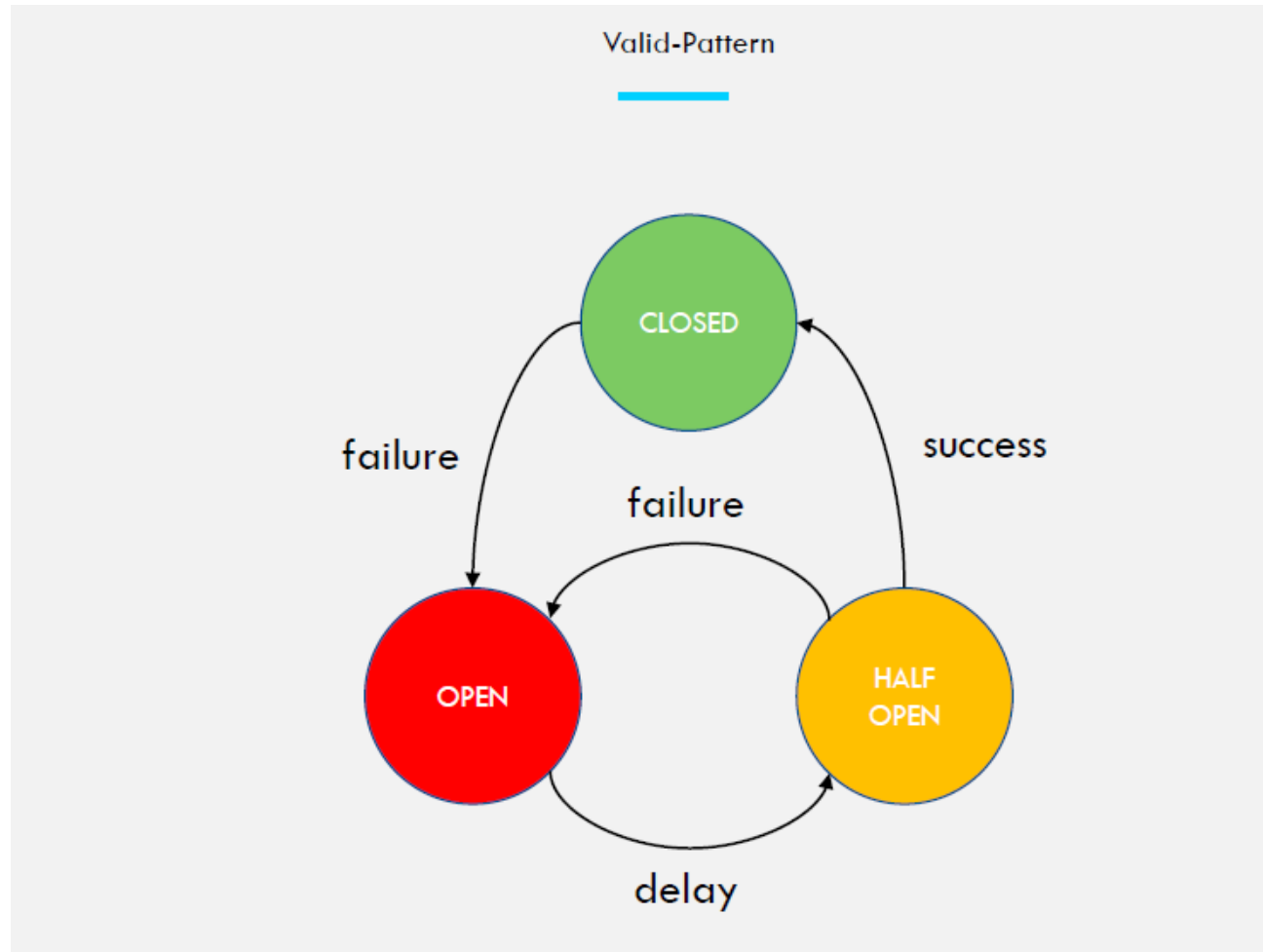




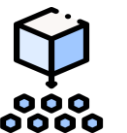
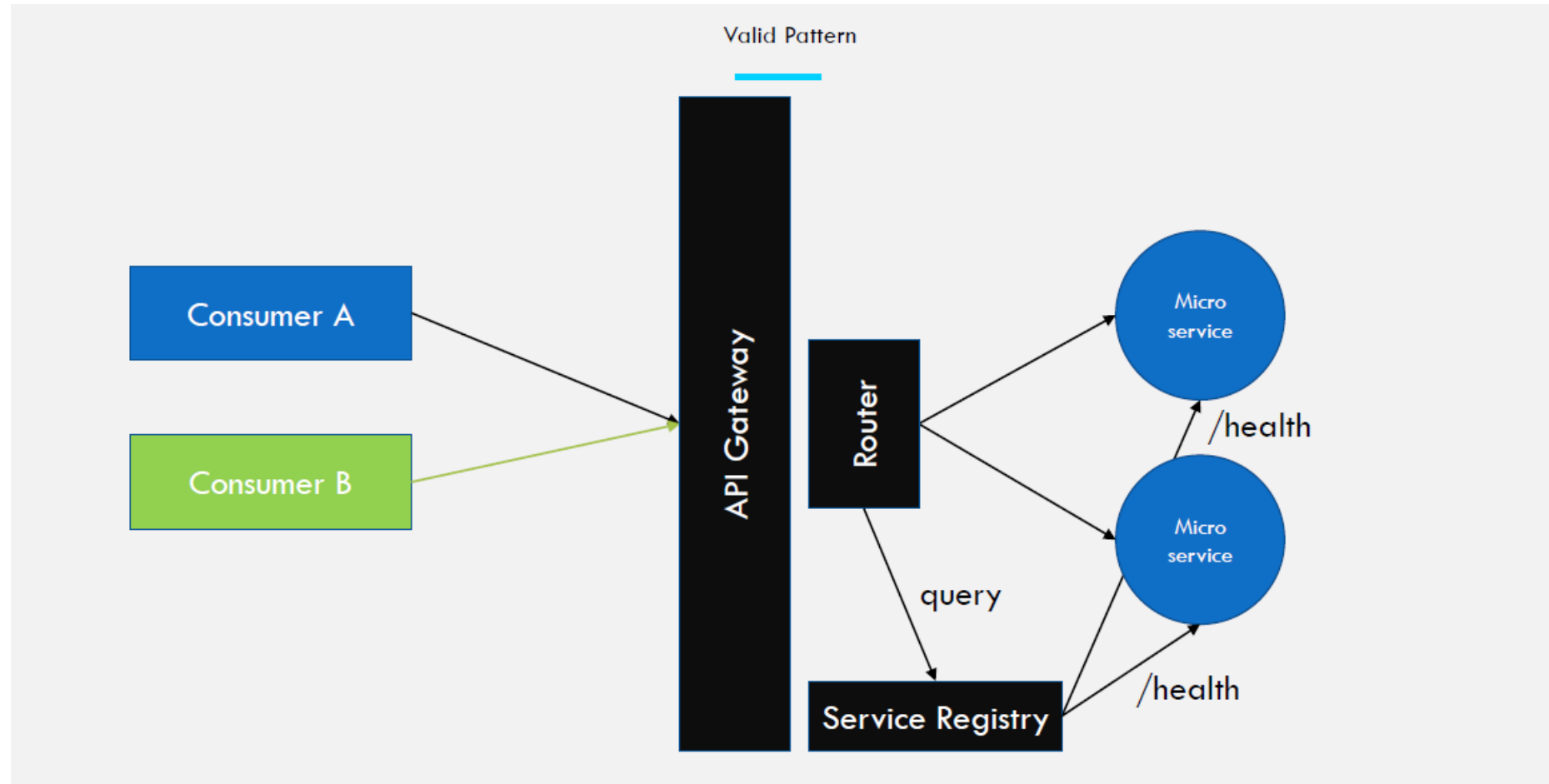
# Retry




# Circuit Breaker



# Health Check API




# Introducción a Spring Boot para microservicios

 Spring Boot es un framework de Java que simplifica la creación de aplicaciones Spring listas para producción con mínima configuración manual. Es ideal para desarrollar microservicios debido a su capacidad de:

- ✓ Auto-configuración.
- ✓ Embedded servers (Tomcat, Jetty).
- ✓ Soporte para arquitecturas distribuidas (Eureka, Feign, Config Server, etc.).



# Microservicios y Spring Cloud

- Spring Cloud Eureka (descubrimiento de servicios)
- Spring Cloud Config Server (configuración centralizada)
- Spring Cloud Gateway / Zuul (API Gateway)
- Spring Boot Actuator (monitoreo)
- Spring Cloud Sleuth + Zipkin (trazabilidad distribuida)
- Spring Security (seguridad/JWT)
-  Docker/Kubernetes (despliegue)



# Casos de uso en la industria.

- Netflix



En el 2008, la ausencia de un punto y coma en un programa, hizo caer el sitio web de Netflix, la recuperación, tomo varias horas. Netflix se hacia mas popular las aplicaciones monolíticas dificultaban escalar era una limitación, en 2009 Netflix movió su arquitectura de monolítica a microservicios a Diciembre del 2011 tenia cientos de microservicios en lugar de gigantescas aplicaciones monolíticas. En diciembre del 2015 Netflix ya manejaba 2 billones de peticiones a sus APIs diariamente.



# Casos de uso en la industria.

- Amazon



En sus inicios, Amazon tenía una aplicación monolítica que dificultaba el desarrollo y despliegue de nuevas funcionalidades. Comenzaron a dividir la aplicación en servicios pequeños y autónomos, cada uno con una responsabilidad clara. Adoptaron el principio de **"You build it, you run it"**: los mismos equipos que desarrollan un servicio son responsables de operarlo en producción. En eventos como re:Invent, se ha mencionado que servicios como Amazon S3 reciben trillones de peticiones mensuales. En 2021, por ejemplo, Lambda ejecutaba billones de invocaciones al mes, lo que da un promedio de más de 10 mil millones por día, solo en ese servicio.



# Cuestionario

- ☐ ¿Que es CI/CD? Describe una buena practica
- ☐ ¿Qué otro beneficio podrías anotas de la adopción de los microservicios?
- ☐ ¿Cuándo seria conveniente aplicar el patrón monolítico para construir un aplicación?

